REMARKS

I. INTRODUCTION

Claims 1-28 remain pending in the present application. No new matter added has been added. In view of the following remarks, it is respectfully submitted that all of the presently pending claims are allowable.

II. THE 35 U.S.C. § 103(a) REJECTIONS SHOULD BE WITHDRAWN

The Examiner has rejected claims 1-26 under 35 U.S.C. § 103(a) as unpatentable over U.S. Patent No. 6,721,948 to Morgan (hereinafter "Morgan") in view of U.S. Patent No. 6,105,098 to Ninose et al. (hereinafter "Ninose").

Morgan describes a task manager that provides for a requesting task (i.e., Process_A) dynamic interrupts of a shared task (i.e., R). (*Morgan*, Abstract). During execution of a shared task, the requesting task stores watchpoints, which contain status data pertaining to the shared task. The watchpoints allow the requesting task to request the task manager to reinitiate performance of the shared task, if the task manager has previously interrupted the performance of the shared task. (*Id.*, col. 5, lines 4-9). Thus, another requesting task (e.g., Process_B) can interrupt Process_A's use of the shared task based on specific conditions (e.g., higher priority) and later on Process_A can resume its use of the share task at the point when the use was interrupted. (*Id.*, col. 5, lines 23-25). Thus, Morgan is directed to an interruption scheme between tasks competing for resources, and not a process for determining whether an interruption should take place.

Ninose discloses a method for managing resources shared among computers. (*Ninose*, col. 2, lines 39-40). To use a shared resource, a program first asks a coupling facility for approval of the use of the shared resource, along with the program identifier. If the program is allowed to use the shared resource, the coupling facility approves of the use of the shared resource to the program, and sets the program identifier in a table in order to keep track of the use status of the shared resources. (*Id.*, col. 6, lines 52-65). When the program terminates the use of the approved shared resource, it tells the coupling facility that the use of the resource is terminated. Upon receipt of the termination of the use of the shared resource from the program, the coupling facility sets an indication that no program is using the resource in the table for keeping track of the use status of the resources. (*Id.*, col. 7, lines 13-19). Then, if a resource request queue is not empty, the coupling facility fetches an entry sequentially from the beginning of the resource request queue, sets a program identifier contained in the entry in the resource use status table, determines from the program identifier the computer on which the program is being executed, and sends approval of the use of the resource to the to the subsequent requesting program. (*Id.*, col. 7, lines 20-25).

In contrast, claim 1 of the present application recites a method for resource stealing by a higher priority task from a holding task comprising "determining whether the holding task has used the resource *since the resource was assigned to the holding task*" and "*releasing the resource when the higher priority task requests to take the resource and the holding task has not used the resource since the resource was assigned to the holding task.*"

Morgan specifically teaches a task manager that allocates resources through interruption. For instance, Morgan states that "in deciding whether or not to interrupt

Process_A's use of R, OS 16 will need to take into considerations various conditions." Since Morgan applies to allocation of resources that are in use through interruption, it does not disclose allocation of resources that are not being used (i.e., without interruption) even though they have been reserved by a holding task. The Examiner acknowledged this shortcoming of Morgan and cited Ninose to cure the deficiency (06/21/04 Office Action, page 3, paragraph 9).

Ninose also fails to disclose a system that manages allocated but unused resources. The present application is directed to managing resources between tasks. The resources are initially assigned to the requesting tasks. However, at some point another task with a higher priority may request the same resource. Even though the resource has been assigned, the second requesting task may still obtain access to the resource if the assigned resource is still not being used by the holding task. (*Specification*, page 6). As stated above, Morgan is only concerned with allocation of currently used resources through interruption without determining if the resource is being used. Conversely, Ninose only teaches allocation of resources that are free and are unassigned to any particular requesting task.

Ninose uses a coupling facility to allocate resources. The coupling facility determines if a request for a specific resource should be granted to the requesting task and also maintains status information on use of a specific resource. If the resource is in use, then the coupling facility would disapprove a request by the task and place the requesting task in queue. (Ninose, col. 5, lines 54-61). Once the resource is free, the next process awaiting in the queue uses the resource. (Id., col. 6, lines 19-22). Thus, Ninose only teaches linear resource allocation based on a queue system using an ENQ set for this purpose. This data structure contains the resource status and the list of programs awaiting for the resource. (*Id.*, col. 4, lines 35-39). In

fact, Ninose teaches away from resource stealing since it describes it as problem with prior art. (*Id.*, col. 4, lines 4-54 (describing delay of execution of process B as subsequent processes take precedence and thus utilize the resource delaying B's execution)).

In contrast, the present application allows for a resource to be utilized by a subsequent process even if the resource is currently reserved by a first process. Unlike Morgan and Ninose, the present application distinguishes between two states of resource management: reservation and use. Neither of the cited references alone or in combination teach the method of resource management as taught by the present invention. Morgan does not distinguish between the usage status of a particular resource and Ninose does not disclose a method for resource stealing. In fact, the orderly distribution of a queue system as disclosed in Ninose is not conducive to the dynamic method of resource allocation as disclosed in the present application.

The Examiner stated that Ninose discloses "determining whether the holding task has used the resource" and "releasing the resource when the higher priority task requests to take the resource and the holding task has not used the resource." (06/21/04 Office Action, page 4, paragraph 9). Ninose utilizes the word "use" to describe the status of a resource (i.e., whether it is free or not). In contrast, the present invention utilizes this word in a different context. According to the present invention a resource is initially assigned or allocated to a requesting task and subsequently the task may "use" the resource to execute itself. Ninose's definition of "use" includes both of those concepts: reservation of resources and their usage for execution. For instance, the process of obtaining a resource utilizes a "resource reservation command (REGE command) 420." (Ninose, col. 9, lines 1-8). Ninose describes the reservation procedure

as usage of the resource by sending REGE command 420 to the coupling facility 100, without distinguishing between reservation and usage, further illustrating the interchangeability of the terms for Ninose. (*Id.*, col. 10, lines 1-15). Furthermore, once a resource is reserved, it is considered to be in use and a corresponding pointer object, RST 270, denoting the resource's status as being used by a particular task is flagged:

> The RST 270 contained in a resource 260 is set to indicate that a program associated with a UID whose value is "a" (a predetermined variable) is using the resource 260. This is referred to as "a reserves a resource 260".

(*Id.*, col. 10, lines 16-19).

Neither Morgan or Ninose include any showing or suggestion of a method for resource allocation comprising the steps of "determining whether the holding task has used the resource *since the resource was assigned to the holding task*" and "*releasing the resource when the higher priority task requests to take the resource and the holding task has not used the resource since the resource was assigned to the holding task,*" as recited in claim 1 of the present application. It is therefore respectfully submitted that claim 1 is not unpatentable over the Morgan in view of Ninose.

Independent claim 2 includes the same limitation as claim 1, i.e., "*determining whether the holding task has executed since the semaphore was assigned to the holding task*" and "*releasing the semaphore when the higher priority task requests to take the semaphore and the holding task has not executes since the semaphore was assigned to the holding task.*" The Examiner made the same assumption that a semaphore is merely a modification of a shared resource. (06/21/04 Office Action, page 5, paragraph 13). Thus, for the reasons described above with reference to claim 1, it is respectfully submitted that claim 2 is not unpatentable over the

Morgan in view of Ninose.

Independent claim 3 also includes the claim language "*determining whether the holding task has executed since the semaphore was assigned to the holding task*" and "*releasing the semaphore when the higher priority task requests to take the semaphore and the holding task has not executes since the semaphore was assigned to the holding task.*" Thus, for the reasons described above with reference to claim 1, it is respectfully submitted that claim 2 is not unpatentable over the Morgan in view of Ninose. Because claims 4-15 depend from and, therefore, include all of the limitations of claim 3, it is respectfully submitted that these claims are also allowable and the rejection of claims 4-15 under 35 U.S.C. § 103(a) should be withdrawn.

Independent claim 16 includes the same limitation as claim 1, i.e., "*setting a variable to indicate that the holding task has not executed since receiving the semaphore when the holding task receives the semaphore . . . determining whether the holding task has executed since receiving the semaphore by testing the variable*" and "*releasing the semaphore when the higher priority task attempts to take the semaphore and the holding task has not executes since the semaphore.*" Thus, for the reasons described above with reference to claim 1, it is respectfully submitted that claim 16 is not unpatentable over the Morgan in view of Ninose.

Independent claim 17 also includes the claim language "*determining whether the holding task has executed since the semaphore was assigned to the holding task*" and "*releasing the semaphore when the higher priority task requests to take the semaphore and the holding task has not executes since the semaphore was assigned to the holding task.*" Thus, for the reasons

described above with reference to claim 1, it is respectfully submitted that claim 17 is not unpatentable over the Morgan in view of Ninose.

Independent claim 18 includes the claim language "*determining whether the holding task has executed since the semaphore was assigned to the holding task*" and "*releasing the semaphore when the higher priority task requests to take the semaphore and the holding task has not executes since the semaphore was assigned to the holding task.*" Thus, for the reasons described above with reference to claim 1, it is respectfully submitted that claim 18 is not unpatentable over the Morgan in view of Ninose.

Independent claim 19 recites a system comprising "a semaphore control mechanism configured to release the semaphore if (a) a first task holds the semaphore, (b) a second task having a higher priority than the first task attempts to take the semaphore, and (c) *when the second task attempts to take the semaphore, the first task has not executed since receiving the semaphore.*" Thus, for the reasons described above with reference to claim 1, it is respectfully submitted that claim 19 is not unpatentable over the Morgan in view of Ninose.

Independent claim 20 recites a system comprising "a semaphore control mechanism configured to release the semaphore if (a) a first task holds the semaphore, (b) a second task having a higher priority than the first task attempts to take the semaphore, and (c) *when the second task attempts to take the semaphore, the first task has not executed since receiving the semaphore.*" Thus, for the reasons described above with reference to claim 1, it is respectfully submitted that claim 20 is not unpatentable over the Morgan in view of Ninose. Because claims 21-25 depend from and, therefore, include all of the limitations of claim 20, it is

respectfully submitted that these claims are also allowable and the rejection of claims 21-25 under 35 U.S.C. § 103(a) should be withdrawn.

Independent claim 26 recites a system comprising "a semaphore control mechanism configured to release the semaphore when the second task attempts to take the semaphore and the first task has not executed since receiving the semaphore." Thus, for the reasons described above with reference to claim 1, it is respectfully submitted that claim 26 is not unpatentable over the Morgan in view of Ninose.

III. THE 35 U.S.C. § 102(e) REJECTIONS SHOULD BE WITHDRAWN

The Examiner has rejected claims 27 and 28 under 35 U.S.C. § 102(e) as anticipated by Ninose.

Independent claim 27 recites a semaphore control block associated with a semaphore comprising "a stealable variable, the stealable variable configured to indicate whether the semaphore can be stolen from the task that presently holds the semaphore with which the semaphore control block is associated. As stated above in regards to claim 1, Ninose does not teach a method for stealing resources and merely creates a queue of tasks requesting a shared resource. Thus, for the reasons described above with reference to claim 1, it is respectfully submitted that claim 27 is not unpatentable over the Morgan in view of Ninose. Because claim 28 depends from and, therefore, include all of the limitations of claim 27, it is respectfully submitted that this claim is also allowable and the rejection of claim 28 under 35 U.S.C. § 102(e) should be withdrawn.

IV. CONCLUSION

In light of the foregoing, the applicants respectfully submit that all of the now pending claims are in condition for allowance. All issues raised by the Examiner having been addressed, an early and favorable action on the merits is earnestly solicited.

Respectfully submitted,

Dated: 10/13/04

By: _____
Michael J. Marcin (Reg. No. 48,198)

Fay Kaplun & Marcin, LLP
150 Broadway, Suite 702
New York, NY 10038
Tel: (212) 619-6000